NASA-TM-112075

[illegible]

BENCHMARKS TO SUPPLANT EXPORT "FPDR" CALCULATIONS

RNR-88-007

David Bailey, NASA Ames Research Center
Eugene Brooks, Lawrence Livermore National Laboratory
Jack Dongarra, Argonne National Laboratory
Ann Hayes, Los Alamos National Laboratory
Michael Heath, Oak Ridge National Laboratory
Gordon Lyon, National Bureau of Standards

Advanced Systems Division
Institute for Computer Sciences and Technology
National Bureau of Standards
Gaithersburg, MD 20899

Sponsored by the
Computer Systems Technology Center,
Office of Export Administration,
Department of Commerce.

U.S. Department of Commerce, C. William Verity, Jr., Secretary

National Bureau of Standards, Ernest Ambler, Director

June 1988

TABLE OF CONTENTS

BENCHMARKS TO SUPPLANT EXPORT "FPDR" CALCULATIONS

David Bailey, NASA Ames Research Center
Eugene Brooks, Lawrence Livermore National Laboratory
Jack Dongarra, Argonne National Laboratory
Ann Hayes, Los Alamos National Laboratory
Michael Heath, Oak Ridge National Laboratory
Gordon Lyon, National Bureau of Standards

On April 28, 1988 a committee met at the National Bureau of Standards to review possible benchmark replacements for the "FPDR," a theoretical estimator of performance for a computer's "floating point data rate." The measure is used in export evaluations. Several recommendations were made in the form of a design study:

1. **An "FPDR" replacement should have four major benchmark components: (i) A test for peak vector performance that is run using a complete system, whether it be a parallel or uniprocessor. A 1000x1000 linear system problem TPP (Toward Peak Performance) from Argonne National Laboratory (ANL) was chosen. This test allows tailored, very low level coding and algorithm substitution. (ii) VECOPS, which are Los Alamos (LANL) floating point FORTRAN vector exercises over various vector lengths. (iii) Five scalar tests from the Livermore FORTRAN Kernels (LFK). (iv) Logical vector exercises, LOGICB, similar to (ii), via NASA.**
2. **Foreign purchasers of multiprocessor machines choose computers which are suited, i.e. architecturally balanced, for the purchasers' purposes. Thus the export evaluation ratings for the suites VECOPS, scalar LFKs, and LOGICB should be the performance on one processor multiplied by the number of processors. This differs from ordinary evaluation practice.**
3. **Exportability performance limits should be set for each suite of tests. Separate limits avoid reducing everything to one performance estimate, a practice dependent upon algorithmic use (which is often unknown).**

Key words: benchmarks; computers; export; 'FPDR'; minimax; scalars; vectors.

No recommendation or endorsement, express or otherwise, is given by the National Bureau of Standards or the sponsor for any illustrative commercial items in the text. Sponsored by the Computer Systems Technology Center, Office of Export Administration, Department of Commerce, Herbert C. Hoover Building, 14th and Consitution Avenue, N.W. Washington, D.C. 20230

Introduction

The Institute for Computer Sciences and Technology, NBS, has been asked by the Office of Export Administration (OEA) to investigate the feasibility of replacing certain formulas used in computer export evaluations with performances on actual benchmark codes. In particular, the "FPDR," which is a theoretical estimator of a computer's "floating point data [processing] rate," has been identified as one criterion that might warrant replacement. The FPDR estimates how fast a computer might perform floating-point evaluations; note, however, that the FPDR includes factors on other capabilities, such as for addressing.

Because modern architectures render application of the FPDR increasingly difficult, there has been OEA interest in export evaluation via actual system performances. It was for this purpose that the advisory committee was formed. There are enough benchmarks available in everyday use that export evaluations for scientific machines might be possible using established components. The committee has been charged with selecting a sound, compact, understandable and reliable set.

The committee met at the National Bureau of Standards, Gaithersburg, on April 28, 1988 after a month of discussion via electronic mail. Scope and agenda for the meeting were deliberately narrow. The first agenda item was the benchmarking of uniprocessor (usually vector) machines for scientific computation; this matched the character of computation in the national laboratories represented by the committee membership. SIMD array processors were not included in this examination.

A second agenda item was parallel processing and its characterization for export control. The result here is specialized for export control and not applicable to the general purchase of large machines.

Export Evaluation versus Everyday Modeling

Export control introduces elements that are not consonant with conventional viewpoints of application modeling. An explanation of the thinking behind this stance will help readers later understand why the scoring is performed as it is.

The most salient factor is that export control has a game-theoretic setting. Machines are seen as being available to potential adversaries. This context supports the idea of "minimax," of minimizing an opponent's maximum gain. In game analysis, gains or payoffs are often displayed in a matrix, with one player's moves on the columns, and the second's via rows. If player A chooses row x, then player B selects column y to minimize A's gain. This is the spirit of the scoring. It follows that anyone not interested in this viewpoint had better not naively use the export evaluations.

Parallelism. The essential elements of parallel computing are assumed to be loosely-coupled (message, or by-value) architectures, and tightly-coupled (shared-memory, or by-reference) multiprocessors. Because loosely-coupled systems have only private memory, one is assured that p copies (for p processors) of a uniprocessor program can run as fast as one copy. And indeed, there are parallel processing algorithms for some problems that do nearly this well. Shared memory machines may suffer memory response degradation with parallel algorithms; the question is how much. Contention can be quite low. (Sometimes, cached values allow shared-memory contention to be ignored.) In the end, it is the algorithms on each architecture that establish whether full performance is available. The committee decided that since nothing would be known about these algorithms, they should be judged ideal. This is simple, fair to manufacturers of both architectures, and consonant with a game-theoretic view.

Everyday Parallelism. For everyday jobs the rosy picture painted above is never true. Loosely coupled systems suffer from message latency (fixed delay) and bandwidth limitations, while shared-memories have memory contention and synchronization costs. Past export evaluations have placed shared-memory performance losses at 25 percent of the number of processors, while loosely-coupled systems have had no such discount. One can argue that this practice is biased against loosely-coupled systems, which can be very hard to program well for some applications. Furthermore, shared-memory with processor caches may have little loss of processor capability, given the right application.

In summary, export evaluation assumes that a machine is sought because it works very well for the intended purpose(s). This may not correspond to everyday experience with algorithms that are but casually matched to an architecture. Export evaluation is different from conventional modeling.

A Benchmark Set

Scalar performance is deemed very important in scientific machines, even though the dominant paradigm in the tests is linear algebra. On some systems, there are important application codes with heavy scalar components that significantly influence any average operations mix; for an example, see Appendix C. Furthermore, no sharp distinction is made as to how a computer actually handles vectors and arrays. Admittedly, the vector processor is often a cost-effective approach to linear algebraic computation, but it is no longer the sole technique. Machines with heterogeneous pipelines or very-wide instructions present alternatives which are also technically effective.

A set of four benchmarks is recommended:

1. TPP, a peak performance test for linear equations.
2. VECOPS, a repertoire of elementary vector floating-point operations over varying vector lengths.
3. Scalar LFK tests, for scalar characterizations.
4. LOGICB, a suite similar to VECOPS, but with logical operations instead of floating-point.

A short discussion of these benchmark sets follows, along with remarks on their scoring for export control. The reader should bear in mind that the methods of scoring, especially for parallel systems, are very specific to the circumstances of export control.

(1) TPP. Toward Peak Performance is Jack Dongarra's benchmark problem of solving a linear system of equations, order 1000. Manufacturers may use any algorithm they wish, although the results are scored for correctness [DON88]. This test should correlate well with theoretical peak performances when it is properly implemented. The test is run and scored for a whole system, so any speedups from parallelism or vector units are included.

(2) VECOPS. These basic vector operations show floating-point capabilities for various vector/scalar (V/s) combinations over adjustable vector lengths. [The ability of a machine to handle shorter vectors

well eases the design of algorithms for it, and thereby makes the machine more productive.] Memory stride is consecutive. One possible modification from the standard LANL set is the inclusion of a second vector index; this keeps cache and compiler mechanisms from defeating the purpose of the synthetic benchmark. Examples of the operations are:

$$V = V + s$$
$$V = V * V$$
$$V = V * V + V * V$$

Vector lengths range typically from 10 to 1000 and will be stated.

(3) Scalar LFK. The scalar features of a modern machine are important; Appendix C shows that 30% of supercomputing at Los Alamos (LANL) can be scalar. The LFK scalar codes are small, so their indicated performance is generally higher than that measured with larger benchmarks. Nonetheless, a good correlation exists among the LFK scalar tests and others [LUB88]. The LFK scalar tests are numbers 5, 11, 17, 19, and 20 [see MCM86, p.14].

(4) LOGICB. This set from NASA is designed to test bit-wise logical computation speed. Such operations, or ones related to them, may constitute a heavy fraction of a vector workload. [At LANL, many operations are vector without being floating-point. See Appendix C.] Essentially LOGICB is VECOPS for bit-wise logical rather than floating-point operations. Appendices A and B have preliminary versions of this benchmark in FORTRAN and C.

Scoring of the Tests

Each of the four benchmarks is scored independently of the others. This is necessary because in export the users' algorithms are often unknown. Only with detailed knowledge of algorithmic operations does an aggregate weighted score make sense.

TPP runs on a whole system, and is thereby fairly straightforward to score. Dongarra [DON88] periodically publishes a list of TPP performances.

The remaining three benchmarks, VECOPS, scalar-LFK and LOGICB, are for uniprocessors. For multiprocessors, one multiplies the single processor result by the multiplicity of units. In everyday modeling, this optimistic estimation would be unacceptable. However, since the end-user's algorithms are probably the largest determinant of performance [COC88], one can only assume that they have been well chosen. Furthermore, the TPP value will provide a cross-check on the overall capability of a parallel system.

Two factors combine to increase the scoring complications of VECOPS, scalar-LFK, and LOGICB. First, testing over a range of vector lengths generates numerous data points. Second, each benchmark is a number of smaller (sub)tests. The combination of both factors presents more numbers than necessary.

Scalar-LFK, most simple of the three, has only subtests (for scalars). Evidence from the past suggests that good production code will tend to perform with the best of the subtests. Hence a scoring suggestion is to average scalar-LFK only over the top three performances.

The numerous vector lengths cause the other problem. Vector lengths can be summarized by Hockney and Jesshope's $n_{1/2}$ and r_∞, as in [HOC81]. $n_{1/2}$ is the smallest vector length that gives one-half of the asymptotic performance r_∞. Each overall set of VECOPS and LOGICB n's and r's can then be averaged in its top half. This should provide a rough estimate of uniprocessor vector capability.

A combination of top-half averaging and $n_{1/2}$ and r_∞ for vectors reduces the measurement points in any evaluation to a more reasonable number.

Independent Testing and Publication

The evaluation of a machine for export can be done by a qualified independent laboratory. The National Bureau of Standards has a long history of assuring the conformance of testing methods, and can assist in establishment of testing services. Furthermore, any testing results should be published widely. This will encourage manufacturers to accurately estimate their machines' capabilities, and at the same time assist technical agents who purchase the machines. Any results must warn, however, that overall evaluations are biased toward export control of scientific computers.

Concluding Remarks on the Design Study

The committee has designed a small set of export tests that assumes little and uses a minimax interpretation of circumstances; the set can be used for uniprocessors, or with provided extensions, for MIMD multiprocessors.

Uniprocessors

Committee benchmark recommendations for uniprocessors are unanimous. The results are for conventional "64-bit" operand machines, usually vector processors.

Appendices A and B have FORTRAN and C code for the LOGICB test. All other pieces are widely available, and should be treated as atomic elements of the benchmark set. LOGICB, however, is a new construct in the recommended design, and is therefore described in detail.

SIMD Machines

SIMD array processors, with their massive operands, are specifically excluded. The characterization of these machines requires special study.

MIMD Multiprocessors

Multiprocessor scoring hinges upon whether "worst case" analysis is appropriate. In the strict minimax interpretation one designs to a worst case outcome, which may well be a realistic indicator of performance on a surprisingly large number of applications (e.g. [GUS88]). But in some areas, the gap between average case and worst case may be too large, and worst cases too rare, to indicate realistic behavior. The Simplex method for linear programming is an example. Consequently, two committee members are uncomfortable with multiprocessor scoring via multiplication by the number of processors. As an alternative, a machine would run a copy of a uniprocessor subtest simultaneously on each of its processors. This corresponds somewhat to a replicated Monte Carlo problem [BEN85], a weakly coupled parallel application. But a majority of the committee concluded that "perfectly parallel" is close enough to reality often enough that export control could use that fact. MIMD scorings remain a topic for future committee discussion.

References

[BEN85] Benyon, Peter B. "Exploiting vector computers by replication." COMPUTER JOUR. 28, 2(1985), 138-141.

[COC88] Cocke, John. "The search for performance in scientific processors." COMM. ACM. 31, 3(March, 1988), 250-253.

[DON88] Dongarra, Jack. "Performance of various computers using standard linear equations software in a FORTRAN environment." Argonne National Laboratory Technical Memorandum No. 23 , April 26, 1988, 27pp.

[GUS88] Gustafson, John L. "Reevaluating Ahmdahl's law." COMM. ACM. 31, 5(May, 1988), 532-533.

[HAY88] Hayes, Ann, et.al. "Supercomputer ranking for export control." Note to committee members from the Computer Research and Applications Group, Los Alamos National Laboratory, circa April, 1988, 2pp.

[HOC81] Hockney, R.W., and Jesshope, C.R. PARALLEL COMPUTERS--Architecture, Programming and Algorithms. Adam Hilger Ltd., (Bristol, 1981).

[LUB88] Lubeck, Olaf M. "Supercomputer performance: The theory, practice and results." Los Alamos National Laboratory Report LA-11204-MS, January, 1988, 58pp.

[MCM86] McMahon, F.H. "The Livermore FORTRAN kernels: a computer test of the numerical performance range." Lawrence Livermore Laboratory report LLNL UCRL-53724, Dec. 1986, available from National Technical Information Service, Springfield, VA., 22161.

Appendix A: The LOGICB Benchmark

```
      PROGRAM LOGICB
C
C This is a vector logical benchmark test proposed to be a part of the
C Office of Export Administration (Department of Commerce)
C benchmark suite. It measures the long vector performance of a system
C in performing full-word bit-wise logical operations. Because the current
C Fortran standard does not provide an efficient means of specifying such
C operations, they are specified in this program using these functions:
C
C   IAND (I1, I2)    64-bit bit-wise "and" of I1 and I2
C   IOR (I1, I2)     64-bit bit-wise "or" of I1 and I2
C   IPAK (I1, I2)    Packs the 32-bit I1 and 32-bit I2 into a 64-bit result
C                    (the hardware format of this packing is immaterial)
C   IUPK1 (I3)       Unpacks the 64-bit I3 to obtain the equivalent of I1
C                    in the definition of IPAK
C   IUPK2 (I3)       Unpacks the 64-bit I3 to obtain the equivalent of I2
C                    in the definition of IPAK
C
C It is anticipated that some revision will be necessary to execute the
C following code on a particular computer system. The revised code need not
C conform to the Fortran-77; indeed, any language that utilizes the full
C power of the hardware may be used, provided the same operations are
C performed.
C
C This version assumes that the INTEGER data type can hold 64 bits of data,
C and that arithmetic operations on positive integers are valid for results
C up to 2^46.
C
C David H. Bailey     May 3, 1988
C
      PARAMETER (N1 = 1024, N2 = 128, NN = N1 * N2)
      DIMENSION IA(N1,N2), IB(N1,N2), IC(N1,N2), ID(N1,N2), IE(N1,N2)
C>
C The following in-line definitions suffice for Cray computers:
C
      IAND (I1, I2) = AND (I1, I2)
      IOR (I1, I2) = OR (I1, I2)
      IPAK (I1, I2) = OR (SHIFTL (I1, 32), I2)
      IUPK1 (I3) = SHIFTR (I3, 32)
      IUPK2 (I3) = AND (I3, 2 ** 32 - 1)
C>
C Fill the arrays with random bits.
C
      CALL RANDL (0, IA)
      CALL RANDL (NN, IA)
      CALL RANDL (NN, IB)
      CALL RANDL (NN, IC)
      CALL RANDL (NN, ID)
      WRITE (6, 1) N1, N2
 1    FORMAT ('VECTOR LOGICAL PERFORMANCE TEST (64 BITS PER WORD)'//
     $ 'ARRAY DIMENSIONS =', 2I8// 'CHECK VALUES:')
C
C Begin timing tests.  The SECOND function is assumed to be the CPU
C timing function on the given computer system.
C
      T10 = SECOND ()
C
      DO 100 J = 1, N2
        DO 100 I = 1, N1
          IE(I,J) = AND (IA(1,1), IB(I,J))
 100  CONTINUE
C
      T11 = SECOND ()
      WRITE (6, '(2I15)') IUPK1 (IE(2,3)), IUPK2 (IE(2,3))
      T20 = SECOND ()
C
      DO 110 J = 1, N2
        DO 110 I = 1, N1
          IE(I,J) = OR (IA(1,1), IB(I,J))
 110  CONTINUE
C
      T21 = SECOND ()
      WRITE (6, '(2I15)') IUPK1 (IE(5,7)), IUPK2 (IE(5,7))
      T30 = SECOND ()
C
      DO 120 J = 1, N2
        DO 120 I = 1, N1
          IE(I,J) = AND (IA(I,J), IB(I,J))
 120  CONTINUE
C
      T31 = SECOND ()
      WRITE (6, '(2I15)') IUPK1 (IE(11,13)), IUPK2 (IE(11,13))
      T40 = SECOND ()
C
      DO 130 J = 1, N2
        DO 130 I = 1, N1
          IE(I,J) = OR (IA(I,J), IB(I,J))
 130  CONTINUE
C
      T41 = SECOND ()
      WRITE (6, '(2I15)') IUPK1 (IE(17,19)), IUPK2 (IE(17,19))
      T50 = SECOND ()
C
      DO 150 J = 1, N2
        DO 150 I = 1, N1
          IE(I,J) = OR (IA(I,J), AND (IA(1,1), IB(I,J)))
 150  CONTINUE
```

```
C
      T51 = SECOND ()
      WRITE (6, '(2I15)') IUPK1 (IE(23,29)), IUPK2 (IE(23,29))
      T60 = SECOND ()
C
      DO 160 J = 1, N2
        DO 160 I = 1, N1
          IE(I,J) = OR (IA(I,J), AND (IB(I,J), IC(I,J)))
 160  CONTINUE
C
      T61 = SECOND ()
      WRITE (6, '(2I15)') IUPK1 (IE(31,37)), IUPK2 (IE(31,37))
      T70 = SECOND ()
C
      DO 170 J = 1, N2
        DO 170 I = 1, N1
          IE(I,J) = OR (AND (IA(I,J), IB(I,J)), AND (IC(I,J), ID(I,J)))
 170  CONTINUE
C
      T71 = SECOND ()
      WRITE (6, '(2I15)') IUPK1 (IE(41,43)), IUPK2 (IE(41,43))
C
C Output results.
C
      R1 = NN * 64. * 1E-6
      R2 = 2. * R1
      R3 = 3. * R1
      WRITE (6, 2) R1 / (T11 - T10), R1 / (T21 - T20),
     $ R1 / (T31 - T30), R1 / (T41 - T40), R2 / (T51 - T50),
     $ R2 / (T61 - T60), R3 / (T71 - T70)
 2    FORMAT (/'PERFORMANCES IN MLOPS:'/ 'V = S a V', F22.2/'V = S o V',
     $ F22.2/ 'V = V a V', F22.2/ 'V = V o V', F22.2/'V = V o (S a V)',
     $ F16.2/ 'V = V o (V a V)', F16.2/ 'V = (V a V) o (V a V)', F10.2)
C
      STOP
      END
C
      SUBROUTINE RANDL (N, IA)
C
C This pseudo-random number generator for vector computers is based on a
C lagged Fibonacci scheme with lags 5 and 17:
C
C   IB(K) = IB(K-5) + IB(K-17) MOD 2^32
C
C The IB array is actually a 128 x 17 array (in order to facilitate
C vector processing). The array IA is obtained from IB.
C
C This version assumes that N is a multiple of 64. Subsequent calls
C generate additional pseudorandom data in a continuous Fibonacci
C sequence. It is initialized by calling with N equal to zero. This
C routine should produce the same pseudorandom sequence on any system
```

```
C that supports 64-bit INTEGER data with arithmetic valid on positive
C integers of size up to 2^46.
C
C David H. Bailey     May 2, 1988
C
      PARAMETER (IBS = 2176, M32 = 2 ** 32 - 1, M = 5 ** 6, L = 2222,
     $  IT0 = 3141592653)
      DIMENSION IA(N)
      COMMON /RANP/ IP1, IP2, IB(IBS)
C>>
C The following in-line definitions suffice for Cray computers:
C
      IAND (I1, I2) = AND (I1, I2)
      IOR (I1, I2) = OR (I1, I2)
      IPAK (I1, I2) = OR (SHIFTL (I1, 32), I2)
      IUPK1 (I3) = SHIFTR (I3, 32)
      IUPK2 (I3) = AND (I3, 2 ** 32 - 1)
C>>
C This section is executed only during initialization.
C
      IF (N .EQ. 0) THEN
        IP1 = 0
        IP2 = 1536
        IB(1) = IT0
C
C Use a linear congruential pseudorandom number generator to initialize IB.
C
        DO 100 I = 2, IBS
          IB(I) = AND (M * IB(I-1) + L, M32)
100     CONTINUE
      ENDIF
C
C For a normal call, use a vectorizable lagged Fibonacci scheme.
C Two 32-bit results are combined to generate one 64-bit output value.
C
      DO 130 K = 0, N - 64, 64
C
C Both of the next two loops are vectorizable.
C>
CDIR$ IVDEP
        DO 110 I = 1, 128
          IB(I+IP1) = AND (IB(I+IP1) + IB(I+IP2), M32)
 110    CONTINUE
C
        DO 120 I = 1, 64
          IA(I+K) = IPAK (IB(I+IP1), IB(I+IP1+64))
 120    CONTINUE
C
        IP1 = IP1 + 128
        IF (IP1 .EQ. IBS) IP1 = 0
        IP2 = IP2 + 128
      IF (IP2 .EQ. IBS) IP2 = 0
  130 CONTINUE
C
      RETURN
      END
```

Appendix B: LOGICB in the Language C

Here is a quick translation of Bailey's logic benchmark to
C, without the initialization for the moment.
All loops were vectorized, and the C version detects
the width of an integer automatically. The initialization
with random bits seemed unneeded, but could be added easily.
The C version is useful as nonstandard language extensions
are required for the FORTRAN version, and several machines
now have good vectorizing C compilers.

Tests:

V = S a V
V = S o V
V = V a V
V = V o V
V = V o (S a V)
V = V o (V a V)
V = (V a V) o (V a V)

E. Brooks
Lawrence Livermore Laboratories
5 May 1988

```
/* An almost literal translation of Bailey's LOGICB to C.
Initialization dropped for the moment.
*/


/* Bits() returns the number of bits in an
int for logical operations.
*/
bits()
{
        int i = 1;
        int nbits = 0;
        while(i != 0) {
                i <<= 1;
                nbits += 1;
        }
        if(nbits < 16 || nbits > 64) {
                printf("Possible problem in bits(), nbits = %d\n", nbits);
        }
        return(nbits);
}

#define N1      1024
#define N2      128

int ia[N1][N2];
int ib[N1][N2];
int ic[N1][N2];
int id[N1][N2];
int ie[N1][N2];

double cputime()
{
        fortran double second();
        double d;
        second(&d);
        return(d);
}

main()
{
        int i, j;
        double t1, t2, t3, t4 , t5, t6, t7, t8;
        int s;

        s = ia[1][1];

        t1 = cputime();

        for(i = 0; i < N1; i += 1) {
                for(j = 0; j < N2; j += 1) {
                        ie[i][j] = s & ib[i][j];
                }
        }

        t2 = cputime();

        for(i = 0; i < N1; i += 1) {
                for(j = 0; j < N2; j += 1) {
                        ie[i][j] = s | ib[i][j];
                }
        }

        t3 = cputime();

        for(i = 0; i < N1; i += 1) {
                for(j = 0; j < N2; j += 1) {
                        ie[i][j] = ia[i][j] & ib[i][j];
                }
        }
        t3 = cputime();

        for(i = 0; i < N1; i += 1) {
                for(j = 0; j < N2; j += 1) {
                        ie[i][j] = ia[i][j] | ib[i][j];
                }
        }

        t4 = cputime();

        for(i = 0; i < N1; i += 1) {
                for(j = 0; j < N2; j += 1) {
                        ie[i][j] = ia[i][j] | (s & ib[i][j]);
                }
        }

        t5 = cputime();

        for(i = 0; i < N1; i += 1) {
                for(j = 0; j < N2; j += 1) {
                        ie[i][j] = ia[i][j] | (s & ib[i][j]);
                }
        }

        t6 = cputime();

        for(i = 0; i < N1; i += 1) {
                for(j = 0; j < N2; j += 1) {
                        ie[i][j] = ia[i][j] | (ia[i][j] & ib[i][j]);
                }
        }

        t7 = cputime();

        for(i = 0; i < N1; i += 1) {
                for(j = 0; j < N2; j += 1) {
                        ie[i][j] = (ia[i][j] & ib[i][j]) | (ic[i][j] & id[i][j]);
                }
        }

        t8 = cputime();

#define BMOPS (N1 * N2 * bits() * 1.0e-6)

        printf("PERFORMANCES IN MLOPS:\n");
        printf("V = S a V: %.2f\n", BMOPS / (t2 - t1));
        printf("V = S o V: %.2f\n", BMOPS / (t3 - t2));
        printf("V = V a V: %.2f\n", BMOPS / (t4 - t3));
        printf("V = V o V: %.2f\n", BMOPS / (t5 - t4));
        printf("V = V o (S a V): %.2f\n", 2.0 * BMOPS / (t6 - t5));
        printf("V = V o (V a V): %.2f\n", 2.0 * BMOPS / (t7 - t6));
        printf("V = (V a V) o (V a V): %.2f\n", 3.0 * BMOPS / (t8 - t7));
}
```

Appendix C: LANL Typical Production Utilization

Elizabeth Williams (now at Supercomputing Research Center) kindly provided a sheet of production run measurements which are summarized briefly below. It is for the XMP416 at LANL, weekend of 2 November, 1987. The statistics are for an epoch of 64 hours, or 27 tera-ticks. All four processors were measured, but the numbers are for a single, average processor.

Percent of Operations

	scalar	vector	
floating point	3	28	=31% f.p.
non-fl. point	27	42	=69% non-f.p.
	=30% scalar	=70% vector	

Continuing with the XMP416, average hardware vector lengths were

(i) integer & logical	51
(ii) floating pt.	46
(iii) mem. ref.	37

Logical operations occur heavily in production Monte Carlo code.